Jini technology, out of the box

By Bill Venners

Throughout the day at the Sun Theater on the convention floor of Java Business Expo conference, two Sun consultants give demos of Jini technology, a new network-centric architecture from Sun that allows clients and services to easily connect and interact over the network.

Mail this article to a friend

As the demo begins, one of these fellows breaks into a fair impersonation of Woody Allen while the other guy holds up a small beige box and says, "This is the lookup server." They then turn their attention to a nearby laptop. "This is a Jini technology-enabled laptop," one of them informs us. With a few key presses on the laptop, a screen shot appears on the large screen hanging above the stage. The screen is filled with what looks like a Web browser displaying a page that is split into two frames, both blank. "This is a Jini technology browser," we are told.

The two consultants then proceed to demonstrate the way in which Jini technology eases network administration by simplifying the addition and subtraction of devices to and from the network. They plug a Jini technology-enabled disk drive into the network. A few seconds later, a disk-drive icon labeled "Quantum Storage" appears in one of the browser frames. They then convince an audience member to plug a printer into the network. A few seconds later an icon labeled "Printer" appears in the browser. Next, they convince another audience member to plug a digital camera into the network. As expected, an icon for the camera appears.

The pair then demonstrate how Jini technology makes it easy for computers and devices hooked to the network to interact with each other. The Woody Allen impersonator clicks on the camera icon, which causes the camera's user interface to appear in the other frame of the browser. Through this user interface, Woody instructs the camera to take pictures of an unsuspecting audience member. One of these pictures is then sent to the printer, which prints it out slowly.

Throughout the demonstration, the presenters attempt to whip up the enthusiasm of the audience by dangling free baseball caps with the Java logo on them in front of the audience, promising them to the audience members most willing to yell out Sun's new marketing mantra: "Anyone, Anywhere, Anytime, on Anything." Jini technology looks really cool, but as the Woody Allen impersonator observes, "No technology will ever be as exciting as a free baseball cap."

Well, that depends on who you ask. Chief Jini technology architect Jim Waldo, who gave a talk Tuesday night at the NYJavaSIG meeting in the Jacob K. Javitz Convention Center, was very enthusiastic -- you might even say evangelistic -- about Jini technology. In his talk, Mr. Waldo explained why he thinks Jini software is a revolutionary, not evolutionary, technology. As he put it, "Jini technology is an attempt to change things very significantly. It's not just trying to wrap something around what already exists. It is trying to change fundamentally the architecture of computing systems."

What is Jini technology?

Jini technology enables the building and deployment of distributed systems that are organized as "federations of services." A *federation* is a set of services that can work together to perform a task. A *service*, the fundamental unit of a federation, is an entity that sits on the network ready to perform some kind of useful function. A service can be anything -- a hardware device, a piece of software, a communications channel, or even a human user. A Jini technology-enabled disk drive, for example, could offer a "storage service." Once a service becomes part of a federation, it then can be used by client programs, other services, or users.

To perform a task, a Jini technology-enabled client (a program, service, or user) enlists the help of services. For example, a client program might upload pictures from the "image storage service" in a digital camera, download the pictures to a "persistent storage service" offered by a disk drive, and send a page of thumbnail-sized versions of the images to a "printing service" of a color printer. In this example, the client program builds a federation consisting of itself, the image storage service, the persistent storage service, and the color printing service. The client and services of this federation worked together to perform the task: to offload and store images from a digital camera and print out a page of thumbnails.

Jini technology consists of a programming model and a runtime infrastructure. The programming model helps you build a distributed system organized in the Jini technology way: as a federation of services and client programs. The runtime infrastructure resides on the network and provides mechanisms for adding, subtracting, locating, and accessing services as the system is used. Services use the runtime infrastructure to make themselves available when they join the network. A client uses the runtime infrastructure to locate and contact desired services. Once the services have been contacted, the client can use the programming model to enlist the help of the services in achieving the client's goals.

Jini technology's runtime infrastructure

The runtime infrastructure of Jini technology resides in two places: in *lookup services* that sit on the network, and in the Jini software-enabled devices themselves. Lookup services are the central organizing mechanism for Jini technology-based systems. When devices are plugged into the network, they register themselves with a lookup service and become part of a federation. When clients wish to locate a service to assist with some task, they consult a lookup service.

Lookup services organize the services they contain into groups. A *group* is simply a set of registered services identified by a string. For example, the `"Printers"` group could be populated by the printing services offered by all the printers on the local network. The `"East Conference Room"` group could be populated by the services offered by all the devices in the East Conference Room (including, potentially, one or more members of the `"Printers"` group). As shown by this example, in which printing services could belong to both `"Printers"` and `"East Conference Room"` groups, a service can be a member of multiple groups. Moreover, multiple lookup services can maintain the same group (can store the group name and its services). This kind of redundancy can help make the Jini technology system more fault tolerant.

If the `"Printers"` group is maintained by multiple lookup services, for example, and one of those lookup services goes off the network, clients will still be able to locate the `"Printers"`" group via a different lookup service.

The runtime infrastructure enables services to register with lookup services through a process called *discovery and join*. Discovery is the process by which a Jini technology-enabled device locates lookup vservices on the network and obtains references to them. Join is the process by which a device registers the services it offers with lookup services.

The discovery process

Discovery works like this: Imagine you have a Jini technology-enabled disk drive capable of offering the service of "persistent storage" to a Jini federation. As soon as you connect the drive to the network, it broadcasts a "presence announcement" by dropping a multicast packet onto a well-known port. Embedded in the presence announcement are two important pieces of information: the IP address and port number where the disk drive can be contacted by a lookup service, and a list of names of groups the device is interested in joining. Assume, for example, that the drive you just plugged into the network declares in its presence announcement packet that it dreams of joining the `"Way Cool Storage Devices"` group.

Lookup services monitor the well-known port for presence announcement packets. When a lookup service receives a presence announcement, it inspects the list of group names contained in the packet. If the lookup service maintains any of those groups, it contacts the sender of the packet directly (using the IP address and port number from the packet) and sends it an RMI stub that will allow it to interact with the lookup service. Thus, in the disk drive example, assume a lookup service that maintains a group named `"Way Cool Storage Devices"` receives the disk drive's announcement packet. Because the announcement packet mentions the disk drive's ambition to become part of the `"Way Cool Storage Devices"` group, the lookup service will contact the originator of the announcement packet -- the disk drive -- directly at the specified IP address and port number. The lookup service will send to the disk drive an object that implements an interface through which the disk drive can register itself, via the join process, as a member of the `"Way Cool Storage Devices"` group.

The join process

Once a device has discovered a lookup service, it can register its own services on that lookup service via the join process. The join process begins when a service connects to a lookup service via the object it received from that lookup service during the discovery process. Through the stub, the service sends information about itself to the lookup service. The lookup service stores the information uploaded from the service and associates that service with the requested group. At that point, the service has joined the group on that lookup service.

The information sent includes an instance of a class that implements a "service interface." It can also include other attributes, including applets that provide graphical user interfaces through which users can directly interact with the service.

The service is identified by the type of the "service interface" uploaded to the lookup service via the join process. Each kind of service is associated with one such Java technology-based interface. The lookup service stores and locates a service based on the type of that interface; clients interact with the service by invoking methods on an object that implements that interface. Thus, a storage service, for example, would upload during the join process an interface that enables clients to interact with the storage service.

The lookup process

Once a service has joined at least one group in a particular lookup service, that service is available for use by clients who query that lookup service. To build a federation of services that will work together to perform some task, a client must locate and enlist the help of the individual services. To find a service, clients interact with lookup servers via a process called *lookup*.

The lookup process begins when a client contacts a lookup service and requests services of a particular type. The type specified in this request is a Java technology-based interface that defines the way in which clients interact with the service being requested. This is the "service interface" that is uploaded from the service to the lookup service during the join process.

The lookup service returns to the client zero to many objects that match the type (that implement the service interface) specified in the client's request. Once a client has an object, it can interact with the service represented by that object. A client interacts with a service by invoking methods on the downloaded object that implements the service interface.

For example, a client program may endeavor to look up a printer service in a lookup service. The client program initiates the lookup process by invoking a method in the object received from the lookup service during the discovery process. In one of the parameters of this method, the client specifies the type (the service interface) by which the desired printer service is known. The lookup server reaches deep within itself, and finds a printer service that matches the request. It sends the object that represents this printer service back to the client, as the return value to the remote method invoked by the client. The client now has an object that implements the Java technology-based interface it requested, an object that serves as a representative of the printer service. With this object, the client can interact directly with the printer service simply by invoking methods on the object.

client/service interaction

The client can interact with a service by invoking methods declared in the service interface on the object that represents the service. In addition, a client can use reflection to look for other interesting methods declared by that object. If the client finds methods that it understands how to use, it can interact with the service by invoking those methods as well, even though those methods aren't part of the service interface.

The object that represents the service can grant the client access to the service in several ways. For example, the object can actually represent the complete service, which is downloaded to the client during lookup and then executed locally. Alternatively, the object can merely serve as a proxy to a remote service. When the client invokes methods on the proxy object, the proxy sends the requests across the network to the service, which does the real work. An in-between approach is also possible. In this case, the local object and a remote service each do part of the work. Proxies that fully or partially implement the service itself are called *smart proxies*.

Note that the protocol used to communicate between a proxy object and the remote service does not need to be understood by the client. This *service protocol* is a private matter decided upon by the service itself. The client can communicate with the service via this private protocol because the service has in effect injected some of its own code (the object that represents the service) into the client's address space. The injected object could be an RMI stub that enables the client to invoke remote methods on an object that exists in the address space of the remote service. Or the injected object could communicate with the service via CORBA, DCOM, or some home-brewed protocol.

Different implementations of the same service interface can use completely different approaches and completely different protocols. A service may use specialized hardware to fulfill client requests, or it may do all its work in software. To the client, a service just looks like a service, regardless of how it is implemented.

The Jini technology programming model

Building a reliable distributed system is difficult because the network is inherently unreliable: servers can crash, traffic can get clogged, wires can be cut. The Jini technology programming model offers a small set of APIs that can help you create reliable distributed systems. Much of the interaction between clients and services during the processes of discover, join, and lookup is built around these APIs, so clients and services will use the Jini technology programming model during those processes at least. But clients and services can also make use of the programming model to do the work for which the federation was assembled in the first place.

The Jini technology programming model consists of three parts: leasing, transactions, and distributed events. Leasing provides a way to manage the lifetimes of distributed objects that can't be managed by the usual rules of garbage collection. In a single address space, the garbage collector can free an object when there are no references to it. But a garbage collector doesn't know if there are any remote references to an object. A lease is a grant of guaranteed access to a remote resource, such as an object, for a specified period of time. It is a guarantee that during the period of the lease, the resource won't be garbage collected away.

For example, if a client wishes to make use of an object in a service, the client can make a lease request to the service that includes a desired lease period. The service can, at its discretion, award a lease to the client. The service gets to decide the duration of the lease, presumably taking the requested time period into account, and communicates that duration back to the client. If the client does not renew the lease before the time period decided upon by the service elapses, the service can assume the object is no longer needed by the client and can discard the object. But so long as the client keeps renewing the lease before it expires (and the service continues to allow the renewal), the service will not garbage collect the object and the object will remain available to the client.

Another aspect of the Jini technology programming model that can help you build reliable distributed systems is transactions. The API that supports transactions enables operations that involve multiple clients and services to either succeed or fail as a unit. If some aspect of the operation managed by a transaction fails, for example, if one of the involved services disappears from the network, the participating parties can be instructed to "roll back" to a known good state.

PvThe third aspect of the programming model that facilitates the building of reliable distributed systems is the distributed event model. This model extends the 1.1 JavaBeans/AWT/Swing event model, which works in a single address space, to distributed systems. Using the Jini technology event model, an object can register itself as a listener interested in events generated by a remote source. When the remote source fires an event, the event will travel across the network to the registered listeners.

Availability of Jini technology

Jini technology is currently available in Development Complete form. You can already download the binaries and the source code from http://java.sun.com/products/jini. An FCS release is expected soon. The first release does not include any security mechanisms beyond what is provided by RMI or the Java platform itself. The next release will contain Access Control Lists. ■

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.